CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

SIGNAL PROCESSING

SPECIAL SECTION:
Embedded Interfacing

RISC vs. DSP

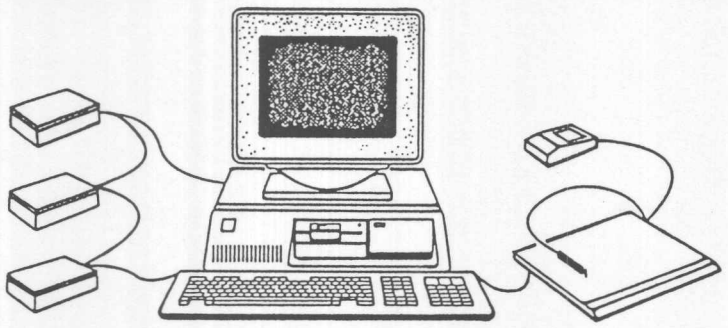
The Photonic Transistor

Instant PC Boards

August/September, 1992 — Issue #28

08
0 74470 75349 0

$3.95 U.S.
$4.95 Canada



ACCESS.bus Specifications

- Speed: 100K bps (400K bps upgrade pending)
- Topology: Bus, tees allowed
- Number of Devices: 14 maximum
- Cable Type: 4-conductor (2 x #26, 2 x #28), shielded
- Cable Length: 8 meters maximum
- Connector: 4-conductor "modular," locking, shielded
- "Hot Plugging"

Figure 1—*ACCESS.bus aims to eliminate the clutter of incompatible cables running out the back of your PC by using a "desktop bus" scheme similar to that used by today's Macintosh computers.*

shielded to minimize RF problems. The specification calls for four-conductor cablee (also shielded) with heavier wire for +5 volts and ground (#26 vs. #28). Unlike the DIN connectors used by ADB and current PC keyboards, the modular ACCESS.bus connector has the advantage of easy orientation. I don't know about you, but I inevitably end up "spinning" DIN connectors a lot even when I can see what I'm doing, not to mention when I'm groping through the all too typical "blind insertion."

Another plus is positive locking. Unlike a phone connector, the dual-release ACCESS.bus connectors seem to make "getting a grip" easier. Notice how the entire connector is streamlined, easing wire routing and minimizing back panel clutter. The design also minimizes the "fishhook" syndrome exhibited by existing connectors (e.g., those DB-25s with the long knurled screws), which seem to get hung up on every possible obstacle as if they were possessed by some mystical attraction.

I²C THE LIGHT

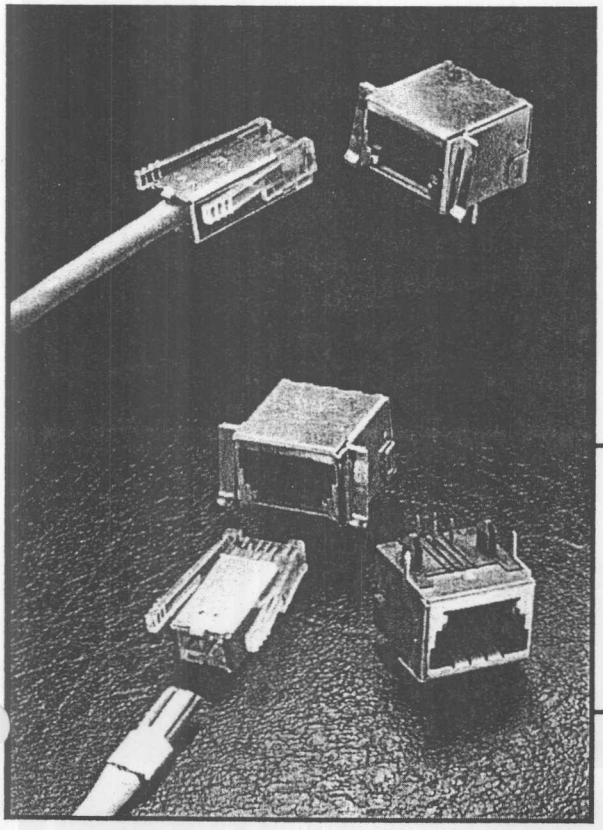
Besides causing grief for users, a multitude of desktop interfaces caused problems for DEC keyboard manufacturing. They had to offer *X* distinct keyboards, where *X* equals the number of popular layouts multiplied by the number of different interfaces. Thus, the seeds were sown for ACCESS.bus.

DEC approached Apple to see if they would consider offering ADB as an open standard. Apparently, Apple's response was something along the lines of "Hey, great idea—not!" so DEC started casting around for alternatives.

Here's where Philips/Signetics enters the picture. To make a long story short, the resulting ACCESS.bus is simply a derivative of that company's Inter-Integrated Circuit (I²C) bus.

I²C was originally designed as kind of a "LAN-in-a-Box," allowing easy connection between processors and interface chips without the bulk and expense of a full-speed parallel bus. Though you may not be familiar with it, I²C is arguably the world's leading LAN because the bus is widely used in high-volume consumer electronics, such as TVs, stereos, and phones. Meanwhile, Philips/Signetics (and others under license) offer a plentiful variety of I²C add-on chips including micros, EEPROMS, real-time clocks, ADC, DAC, and so forth.

I²C is surprisingly sophisticated, despite its low chip cost, simple wiring, and a simple clocked serial port basis where data (SDA) is sampled when the clock (SCL) is high. On top of the basic communication mechanism, I²C layers a message format consisting of the destination address, a read/write flag, and the data framed by start and stop conditions. Furthermore, each byte transferred requires an ACK



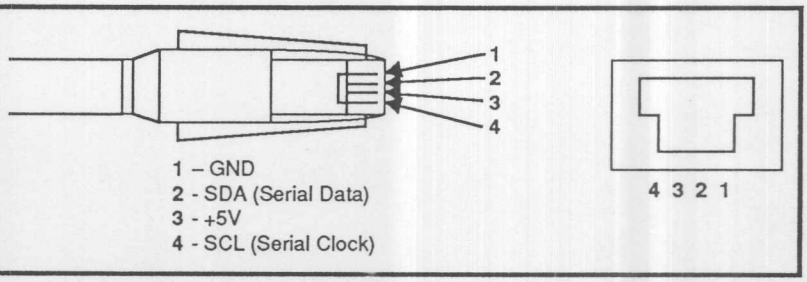


Figure 2—*Based on I²C, ACCESS.bus uses a simple four-wire interface that provides not only a data channel, but also power to peripherals. The proposed modular connector locks in place and eliminates orientation confusion.*

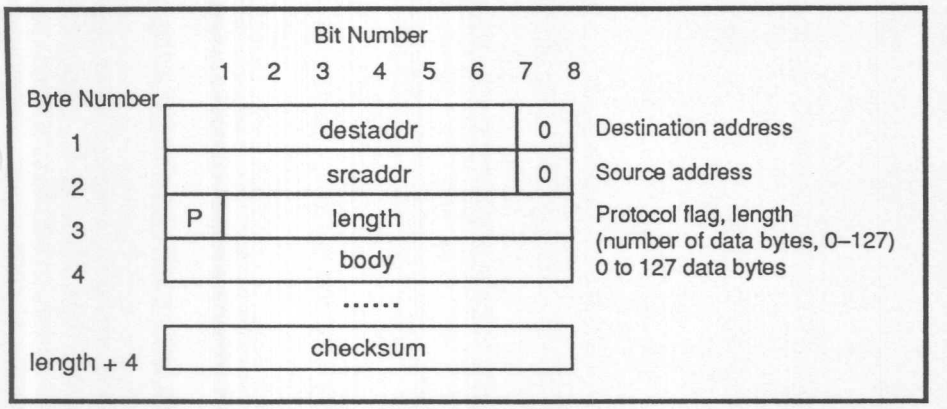


Figure 4—*The standard ACCESS.bus message packet is based on I²C, but includess additional information. The packet consists of a destination address, source address, length, data bytes, and a simple checksum.*

devices. This restriction is reasonable because having your keyboard and mouse talking to each other behind your back seems rather risky.

WHO'S ON FIRST

ACCESS.bus, like all LANs, faces the classic problem of uniquely identifying each node. At power up (or in response to a `RESET()` command) each ACCESS.bus device reverts to the default address. Next, the computer sends an `Identification Request` command to the default address (therefore, to all devices on the bus).

At this point, every device will attempt to reply with their 32-bit ID. The ID can be a unique serial number embedded in each device's ROM. However, because this practice adds cost, the protocol also allows devices to generate their own ID, typically via a counter cleared at reset and incremented by the device's internal clock. The result is two of the same devices will usually come up with a different ID thanks to a slight difference in circuit timing.

That all the devices are trying to respond at once is resolved by the previously mentioned multimaster arbitration mechanism. As each ID message gets through, the computer sends an `Assign Address` command based on the ID. Once the device receives this command, it will assign itself the address specified. Once a device knows its address, it can commence sending and receiving data.

You probably have noticed this procedure has a small, but potentially fatal, loophole: the rare case that two devices report the same ID in the `Identification Request` phase. The subsequent `Assign Address` command will assign both devices the same ACCESS.bus address, which will surely cause problems.

To cinch this loophole shut, ACCESS.bus adopts one final trick. After receiving an address, but prior to first data transmission, each device sends a reset message to its own address. The device sending the message is not itself reset, but any other devices at the same address are. Those devices that are reset will reenter initialization phases in order to receive new addresses.

TIMING IS EVERYTHING

The proponents of ACCESS.bus are careful to keep reminding us that it is mainly designed for low-speed and low-frequency (i.e., human) input devices. There is a danger of users and suppliers of other I/O devices boarding the bus without a ticket.

Witness the case with the PC printer port that has been hooked to just about every kind of I/O device including hard disks. The problem is that hooking high-speed block I/O devices could result in a compromised response. Devices like keyboards or mice may not generate a lot of data, but users won't be happy if they don't perceive these devices' responses as instantaneous.

To this end, the specification imposes a number of limits on the amount of traffic or delays any device can impose. For instance, a so-called noninteractive device like a laser printer can only occupy the bus for 5 ms at a time, which limits the maxi-

Computer-to-Device Messages	**Purpose**
Reset()	Force device to power-up state and default I^2C address.
Identification Request()	Ask device for its "identification string."
Assign Address (ID string, new addr)	Tell device with matching "identification string" to change its address to "new address."
Capabilities Request (offset)	Ask device to send the fragment of its capabilities information that starts at "offset."
Device-to-Computer Messages	
Attention(status)	Inform computer that a device has finished its power-up/reset test and needs to be configured; "status" shall be the test result.
Identification Reply(ID string)	Reply to Identification Request with device's unique "identification string."
Capabilities Reply(offset, data frag)	Reply to Capabilities Request with "data fragment," a fragment of the device's capabilities string; the computer uses "offset" to reassemble the fragments.
Interface Error ()	Invalid checksum or premature end of message detected.

Figure 5—*An ACCESS.bus message may be either data or status/control. Eight status/control messages are defined, four in each direction.*